THEORETICAL REPORT IFS-TR-027

THE NEED FOR REGULARISATION WHEN TRAINING FEED-FORWARD NETWORKS

ABSTRACT

One of the most important considerations when training Multi-Layer Perceptrons or other types of supervised feed-forward networks using least squares techniques is the need to control the *complexity* of the mapping being performed by the networks. This is often referred to as *regularising* the networks (a term borrowed from the statistics literature).

The reason controlling complexity is so important is that it will critically affect the generalisation performance of the network, that is, the performance of the network on unseen data. In this report, we shall examine the problems associated with training neural networks without regularisation and then give an overview of the different strategies that may be adopted to tackle this problem.

REPORT

BACKGROUND

A feed-forward neural network trained on a set of example data using a least squares misfit function is essentially a statistical model whose parameters (the weights of the network) are found using Maximum Likelihood (ML) estimation. However, because of the large number of parameters present in a neural network model (for instance, a 30-16-1 MLP will contain at least 496 separate parameters) there is a significant danger of the estimation of the parameters being extremely *biased.*

We shall illustrate this with an example. Consider the points shown in Figure 1. The points clearly lie approximately on a line. If we were to fit a least squares model to the data using a linear model (having two parameters) we would obtain the line shown in Figure 2. This would appear to be a relatively *good* fit to the data in that the line passes closely to each of the sample points. We would account for the fact that the fit is not exact by assuming some noise on the data.

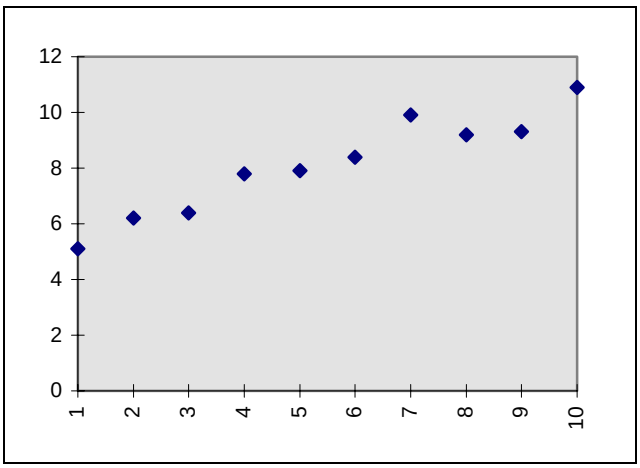

Figure 1: Sample data set

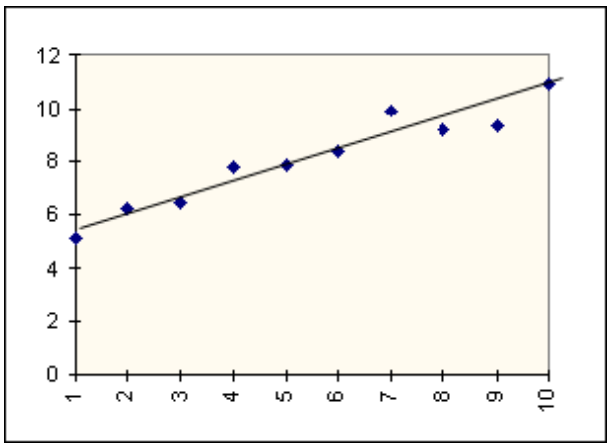

Figure 2: Linear regression line through the data set

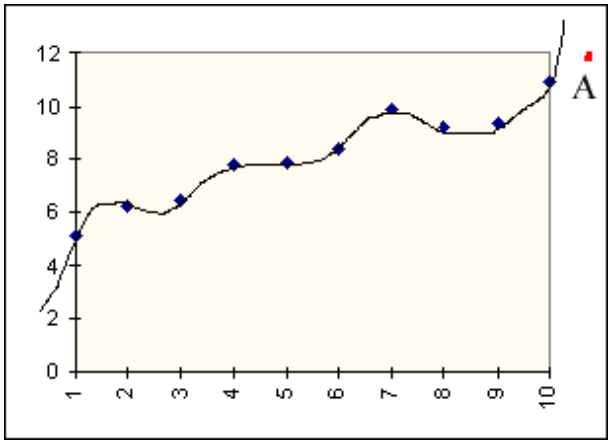

Figure 3: High-order polynomial fit of data set

On the other hand if we were to fit, say, an order 50 polynomial to the data (having 51 parameters), we may obtain the model shown in Figure 3. **In terms of least square error** this is actually a much *better* model than the straight line as the model fits all of the training data points exactly. However, intuitively the model seems poor as, if we attempt to extrapolate the model to areas of the graph where there are no training points (for example the point marked A), the model appears somewhat unlikely.

This is, in fact, a well known problem associated with the use of Maximum Likelihood techniques with models containing a large number of parameters. The problem manifests itself with regard to neural network training, as the network training very well on its training set (i.e. fitting the training points extremely well) but then performing very poorly on *unseen* data. We would say, in this case, that the network has poor *generalisation.*

The problem of *over-fitting* has been known for a long time in the neural network community. There have been a number of suggestions of methods to deal with the problem over the last decade. In this report we shall examine some of these approaches.

APPROACHES TO OVER-FITTING

Cross-validation or *Early-Stopping*

One of the most popular techniques used to control over-fitting in neural networks has been the method of cross-validation or early-stopping. Briefly, this technique operates in the following manner.

The total set of training vectors for the network is divided into two disjoint sets, a *training* set and a *validation* set. During training, the total errors of classifying training vectors for both sets are determined. However, at the end of a training epoch, when it becomes time to update the weights of the network using the partial derivatives of error with respect to each individual weight, **only the training set patterns are used**. This means that the network is effectively learning only from the training set. The error of classifying the validation set should then give an indication of the expected performance of the network on unseen data.

During training, both the training set and validation set errors are monitored. Typically, both of the errors will initially decrease at the beginning of training. However, after a certain period of time, the validation error will start to rise whilst the training error continues falling. This is the point at which training should be halted as it represents the best performance of the network on unseen data.

The main advantage of early-stopping is that it is a relatively simple technique to use and for many problems may be sufficient. However, it relies on a number of assumptions. For more details about the algorithm see Theoretical Report IFS-TR-025.

Weight Decay

Weight decay relies on the principal that the larger the absolute value of the weights of a network, the more complex the mapping between inputs and outputs will be. Therefore, if we wish to control the complexity of the network whilst training, we should penalise models that have high absolute values of weights. In practice this is done by adding an additional term to the usual sum of squares error function for the network, i.e.

$$E(\underline{\omega}) = \frac{1}{2}\sum_{i=1}^{T}\left(y(\underline{x}, \underline{\omega}) - t_i\right)^2 + \lambda \sum_{i=1}^{W} \omega_i^2$$

The second term in the above equation effectively penalises the sum of squares of the weights in the network. This will encourage final networks to have small weights. The balance between the need for small weights and the need to fit the training data is adjusted using the parameter λ.

Weight decay in fact has its roots in the statistical technique of *Ridge Regression* used for fitting linear models. The key factor in its success lies in the choice of a suitable value for λ. For more details concerning weight decay see Theoretical Report IFS-TR-028.

Bayesian Techniques

The weight decay technique above relies on the fact that lower model complexity can be achieved by keeping the weights small. Another approach to regularisation is to use prior distributions for the weights to limit their complexity. In fact, the Bayesian approach leads to a similar technique to that of weight decay but also allows us to reason more effectively about the balancing parameter λ. For details about the use of Bayesian techniques applied to neural network training see Theoretical Reports IFS-TR-029 and IFS-TR-030.

Architecture Pruning / Selection

A number of techniques relying on selecting the appropriate network architecture have been suggested. These range from simple search strategies for the optimal number of hidden nodes to use, to more elaborate techniques involving the pruning or growing of nodes during training. Unfortunately, many of these technique suffer from being somewhat *ad hoc* in nature and have been superseded by more theoretically based statistical and Bayesian techniques

SUMMARY

Controlling the complexity of the neural network whilst training is an extremely important task. This is particularly true for the case of financial forecasting where we are faced with a very noisy set of data with, at best, a tiny predictable element. Because of the problems of non-stationarity in financial time series, we are often also faced with the problem of relatively small training sets as we can only reliably use historical data from the recent past to train the networks.

Amber supports a number of the techniques described above for controlling complexity. For more details see the technical reports listed below and the *Amber Reference Guide.*

Author: Darren Toulson
Revision: v 1.00 4 January, 1997
See ALSO: Theoretical Reports IFS-TR-23, IFS-TR-028 to IFS-TR-030 , The Amber Reference Guide